Lab01 ~ Bags of integers!

This lab is very simple and intended to get you back into the swing of Java programming.

Recall that the number of elements an array has must be specified when you create it and cannot be changed later. However, in many situations, it is impossible to know in advance how many elements will actually be needed. In such cases, the only solution is to allocate an array large enough to handle the worst-case scenario and then to use a subset of its elements as needed. There are various ways to specify the subset, the most common of which is to store data values sequentially from the beginning of the array and to maintain a count of the number of such data values, thus making it easy to ensure that only those elements that contain valid data are processed. The following exercises ask you to create a class that can hold a varying number of integer values and to use this to solve a number of simple problems.

(a) Design and implement a class, *IntBag*, that allows a variable sized collection of integer values to be stored. Your class should have two properties, an int array called *bag* that will hold the values of the collection, and an int value, *valid*, that says how many of the elements of *bag* actually contain values in the collection. The values themselves must be stored in the first *valid* elements of *bag*. The class should have two constructors, one that creates an empty collection (with room for up to 100 elements) and another that takes the desired maximum number of elements as a parameter. Provide methods to *add* a value to the end of the collection, and to *add* a value at a particular index location *i* within it (moving other values "up" to make room and checking *i* is within bounds). Also, write methods to *remove* the value at a given index (moving other values back "down" the array) and another to test whether the collection *contains* a given value or not. Provide a *toString* method that returns a String representation of the collection, a method, *size*, that returns the number of values currently in the collection, and finally a method that will allow you to *get* the value at location *i* within the collection.

(b) Design & implement a program to efficiently compute & display the first 100 prime numbers by making use of the fact that a number is prime if and only if it is not divisible by any prime less than itself. Use an instance of your *IntBag* class to keep a collection of primes found so far. Initially it should contain only the number 2 (the first prime). Generate and check candidate values sequentially from 3 onwards, adding any value found to be prime to the collection until the number of values in the collection is 100. Efficiently check whether a candidate value is prime by attempting to divide it by each of the values in the collection.

(c) Add a method, *findAll*, to your *IntBag* class which returns the locations (indexes) of all instances of a given value in the collection. To demonstrate and test this method, write a program that presents the user with a menu having the following options (which can be selected in any order by typing the corresponding number):

1. Create a new empty collection with a specified maximum capacity (any previous values are lost!)
2. Read a set of positive values into the collection (use a negative value to indicate all the values have been entered.)
3. Print the collection of values.
4. Add a value to the collection of values at a specified location
5. Remove the value at a specified location from the collection of values
6. Read a single test value.
7. Compute the set of locations of the test value within the collection* *(see note below)*.
8. Print the set of locations.
9. Quit the program.

** note about menu options 6, 7 & 8:*
Assume your collection of values contains 3, 5, 4, 5, 2, 7, 5, 2
Given a test value of 5 (obtained using menu option 6)
the new method, *findAll* (called from menu option 7), should return the set of locations of 5 in the collection (in this case 1, 3, 6--assuming locations are numbered from zero.) If the test value were 2, the method should return 4, 7. Of course, option 7 will only do the computation. To see the results (the set of locations) the user must select option 8.
